```
// Verilog HDL for Icache, ic_decompression _behavioral

`define FIXED_FORMAT 10'b1010101010
`define NOP_OPERATION 42'b0

module    ic_decompression (data512, pc,
                        operation4, operation3, operation2, operation1,
                              operation0, format_out, first_word,
                        format_ctrl0,
                        reissue1, stall_in, freeze, reset, clk);

input [511:0] data512;
input [31:0] pc;
output [43:0] operation0, operation1, operation2, operation3, operation4;
reg [43:0] operation0, operation1, operation2, operation3, operation4;
output [31:0] first_word;
reg [31:0] first_word;
input format_ctrl0;
input reissue1, freeze, reset, clk;
wire [9:0] format_out1;
output [9:0] format_out;
reg [9:0] format_outA, format_p;
input stall_in;

// local
reg [9:0] format_out0;
reg [31:0] pc_p;
reg format_ctrl;

reg [9:0] format;
reg used0, used1, used2, used3, used4;
reg [1:0] size0, size1, size2, size3, size4;

reg [511:0] data512shift;
reg [255:0] data256;
reg [2:0] pos1,  pos2,  pos3,  pos4, pos_ext;

reg [25:0] fix0, fix1, fix2, fix3, fix4;
reg [79:0] extension0, extension1, extension2, extension3, extension4;
reg [15:0] ext0, ext1, ext2, ext3, ext4;
reg reset_p;

// format pipe
always @(posedge clk)
begin
reset_p <= reset;

if (reset_p)
begin
      // force NOP operations on instructions
      operation0[42] <= `ONE;
      operation0[43] <= `ONE;
      operation1[42] <= `ONE;
      operation1[43] <= `ONE;
      operation2[42] <= `ONE;
      operation2[43] <= `ONE;
      operation3[42] <= `ONE;
      operation3[43] <= `ONE;
      operation4[42] <= `ONE;
      operation4[43] <= `ONE;
end
else if (~stall_in)
begin

pc_p <= pc;
format_ctrl <= format_ctrl0;
```

```
if (format_ctrl)
   format = `FIXED_FORMAT;
else
   format = format_out;

used0 = ~(format[1] & format[0]);
size0 = used0 ? format[1:0] : 2'b0;

used1 = ~(format[3] & format[2]);
size1 = used1 ? format[3:2] : 2'b0;

used2 = ~(format[5] & format[4]);
size2 = used2 ? format[5:4] : 2'b0;

used3 = ~(format[7] & format[6]);
size3 = used3 ? format[7:6] : 2'b0;

used4 = ~(format[9] & format[8]);
size4 = used4 ? format[9:8] : 2'b0;

// first alignment stage
// rotate the 512 bit word right over a distance between 0 and 64 byte
//
// the rotate is implemented here by swapping the left and right word if the
// distance is more than 32 byte and then perform a right shift over a distance
between
// 0 and 32 byte.

data512shift = pc_p[5] ?
           {data512[255:0], data512[511:256]} >>  {pc_p[4:0], 3'b0} :
           data512 >> {pc_p[4:0], 3'b0};

data256 = data512shift[255:0];

// extract format bits
format_out0 = data256[9:0];

// access first word
first_word <= data256[31:0];

// Notes: - the value for pos_ext==0 is don't care
//        - for values pos_ext < 5 , less than 80 bits are needed

// determine the position of issue slots

//pos0 = 0;
pos1 = used0;
pos2 = used0 + used1;
pos3 = used0 + used1 + used2;
pos4 = used0 + used1 + used2 + used3;

// mux the fixed part of issue slots, combine the 24-bit part and the 2-bit part

fix0 = used0 ? {data256[15:14], data256[(0+1)*24+15 : 0*24+16]} : `NOP_OPERATION
;

fix1 = used1 ?
                (
                  pos1 == 0 ? {data256[15:14], data256[(0+1)*24+15 : 0*24+16]} :
                              {data256[13:12], data256[(1+1)*24+15 : 1*24+16]}
                )
               : `NOP_OPERATION;

fix2 = used2 ?
                (
```

```
                  pos2 == 0 ? {data256[15:14], data256[(0+1)*24+15 : 0*24+16]} :
                  pos2 == 1 ? {data256[13:12], data256[(1+1)*24+15 : 1*24+16]} :
                              {data256[11:10], data256[(2+1)*24+15 : 2*24+16]}
                )
              : 'NOP_OPERATION;

fix3 = used3 ?
                (
                  pos3 == 0 ? {data256[15:14], data256[(0+1)*24+15 : 0*24+16]} :
                  pos3 == 1 ? {data256[13:12], data256[(1+1)*24+15 : 1*24+16]} :
                  pos3 == 2 ? {data256[11:10], data256[(2+1)*24+15 : 2*24+16]} :
                              {data256[95:94], data256[(0+1)*24+95 : 0*24+96]}
                )
              : 'NOP_OPERATION;

fix4 = used4 ?
                (
                  pos4 == 0 ? {data256[15:14], data256[(0+1)*24+15 : 0*24+16]} :
                  pos4 == 1 ? {data256[13:12], data256[(1+1)*24+15 : 1*24+16]} :
                  pos4 == 2 ? {data256[11:10], data256[(2+1)*24+15 : 2*24+16]} :
                  pos4 == 3 ? {data256[95:94], data256[(0+1)*24+95 : 0*24+96]} :
                              {data256[93:92], data256[(1+1)*24+95 : 1*24+96]}
                )
              : 'NOP_OPERATION;

// determine the position of the extension part
pos_ext = used0 + used1 + used2 + used3 + used4;

// determine the extension
extension0 =
                  pos_ext == 0 ? data256[0*24+80-1+16 : 0*24+16] :
                  pos_ext == 1 ? data256[1*24+80-1+16 : 1*24+16] :
                  pos_ext == 2 ? data256[2*24+80-1+16 : 2*24+16] :
                  pos_ext == 3 ? data256[3*24+80-1+16 : 3*24+16] :
                  pos_ext == 4 ? data256[1*24+80-1+96 : 1*24+96] :
                                 data256[2*24+80-1+96 : 2*24+96];

// shift the Extension part
extension1 = extension0 >> {size0 , 3'b0};
extension2 = extension1 >> {size1 , 3'b0};
extension3 = extension2 >> {size2 , 3'b0};
extension4 = extension3 >> {size3 , 3'b0};

ext0 = extension0[15:0];
ext1 = extension1[15:0];
ext2 = extension2[15:0];
ext3 = extension3[15:0];
ext4 = extension4[15:0];

// assemble instruction
//operation0 <= {format_out0[1:0], ext0, fix0};
//operation1 <= {format_out0[3:2], ext1, fix1};
//operation2 <= {format_out0[5:4], ext2, fix2};
//operation3 <= {format_out0[7:6], ext3, fix3};
//operation4 <= {format_out0[9:8], ext4, fix4};
operation0 <= {format[1:0], ext0, fix0};
operation1 <= {format[3:2], ext1, fix1};
operation2 <= {format[5:4], ext2, fix2};
operation3 <= {format[7:6], ext3, fix3};
operation4 <= {format[9:8], ext4, fix4};

if (~freeze | reissue1)
begin
   format_outA <= format_out0;
end
```

```
if (~freeze)
begin
  format_p <= format_outA;
end


end
end

assign format_out = reissue1 ? format_p : format_outA;
assign format_out1 = format;


endmodule
```